CS 535 Object-Oriented Programming & Design
Fall Semester, 2003
Doc 14 Abstract Classes, Inheritance & Testing

References

Object-Oriented Design Heuristics,Riel

Reading

Object-Oriented Design Heuristics, Chapter 2.

Abstract Classes

Abstract class – a class not used to create instances of itself

Concrete class – a class that we do create direct instances of

Why Abstract Classes

- Define interface for subclasses
- Define methods for subclasses
- Define a type
- Hide the existence of concrete subclasses

Defining an Abstract Class

Some languages have special syntax for abstract classes

```
public abstract class NoObjects {
  public void aFunction() {
    System.out.println( "Hi Mom" );
  }
  public abstract void subClassMustImplement( int foo );
}
```

Smalltalk does not have special syntax for abstract classes

- Mark methods as abstract with “self subclassResponsibility”

```
Collection>>do: aBlock
  self subclassResponsibility
```

- Indicate class is abstract in class comment

What does self subclassResponsibility do?

Inform reader

- Method is abstract
- Concrete subclasses need to implement the method

Raises an exception when executed to indicate

- Subclass did not implement an abstract method
- Created an instance of an abstract class

How to Prohibit Instances of Abstract Class

Documentation is normally enough

Implement new so it throws an exception

```
Stream class>>new
   "Provide an error notification that Streams are not created
   using this message."

   self error: ('Streams are created with on: and with:')
```

How do subclass objects get created?

PositionableStream is a subclass of Stream

```
PositionableStream class>>on: aCollection
   ^super new on: aCollection "Throws and Exception"
```

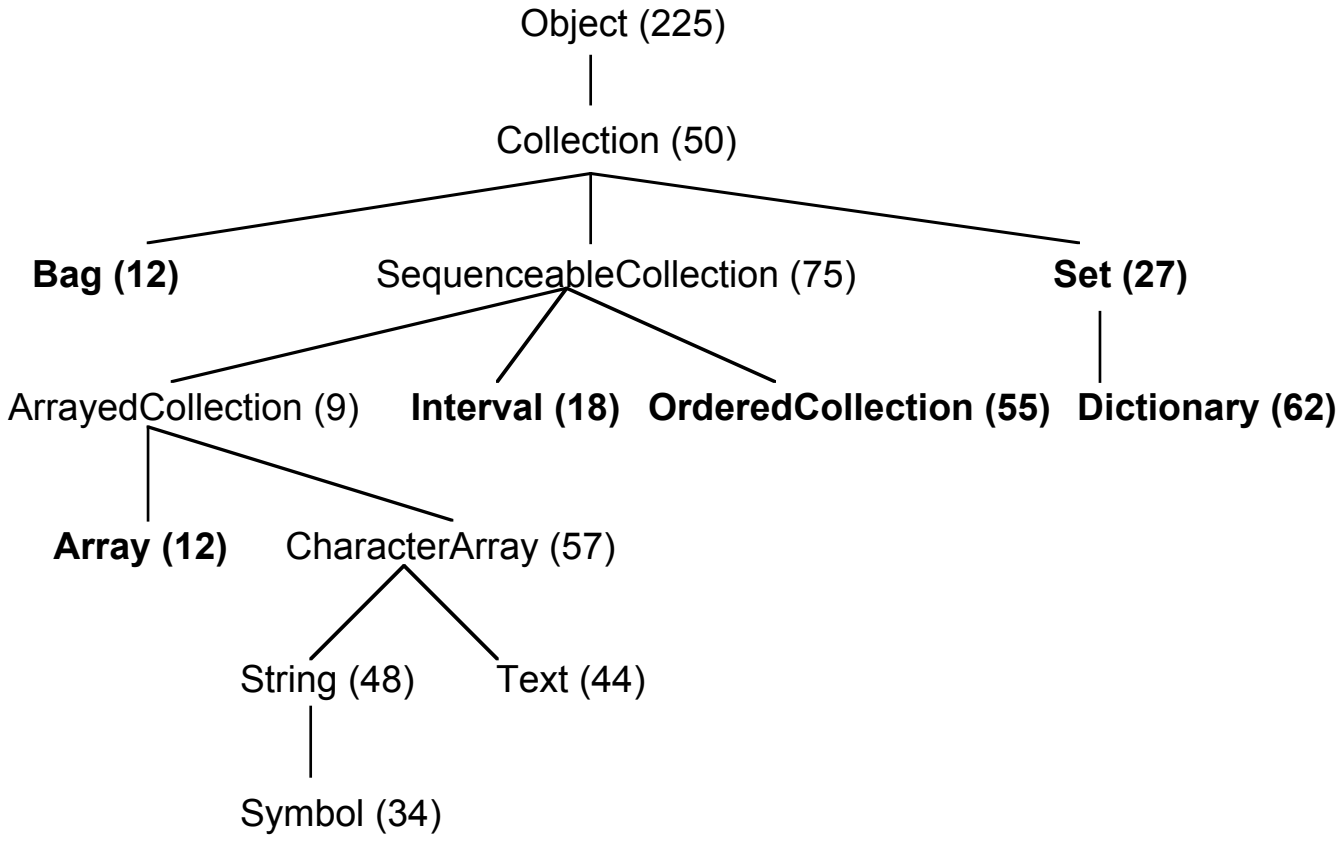
Use basicNew

```
PositionableStream class>>on: aCollection
   ^self basicNew on: aCollection
```

basicNew

- Does the same thing as new
- Is used to get around super class's new method
- Only used in class instance creation methods

Some Collection Classes



(N) indicates number of methods defined the class

Bold indicates concrete classes

Abstract Classes and Data

Abstract classes commonly do not have instance variables

How can they implement methods?

Identify a core set of abstract operations

Implement other methods using core methods

Collection Class

Collection does not have any instance variables

Collection implements
- collect:
- detect:
- detect:ifNone:
- do:separatedBy:
- fold:
- groupedBy:
- inject:into:
- reject:
- select:

All are defined in terms of do:

```
Collection>>detect: aBlock ifNone: exceptionBlock

   self do: [:each | (aBlock value: each) ifTrue: [^each]].
   ^exceptionBlock value


Collection>>do: aBlock
   self subclassResponsibility
```

Subclass just implements do: the rest will work

All the above enumerations work on your BinarySearchTree

Abstract Classes, Types and Hinges

Tagging (declaring) a variable to be an Abstract class instance

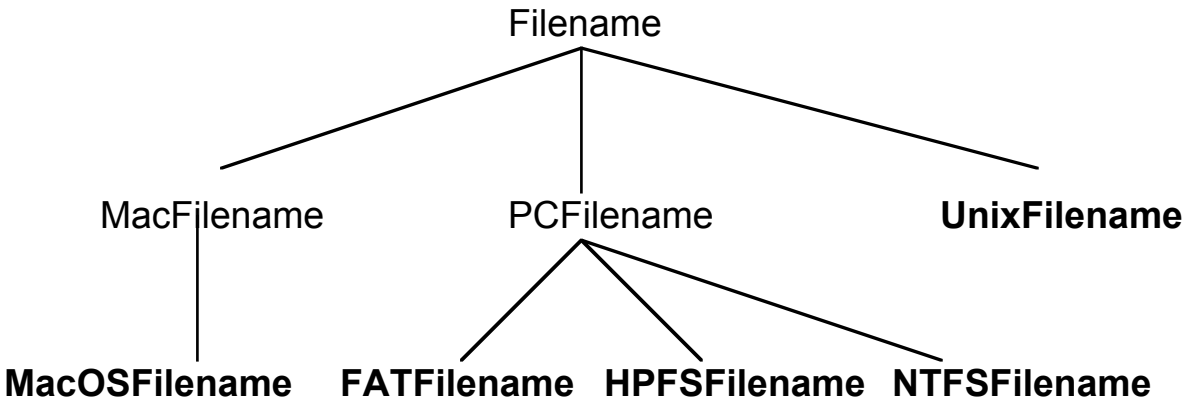
- Indicates which operations are allowed on the variable
- Allows any subclass to be used in the variable
- Provides flexibility particularly in languages with static type checking

```
SomeClass>>foo: aCollection
  ^aCollection fold: [:a :b | a max: b].

public class SomeClass {
  public int foo(Collection a) { blah}
}

public class Resticted {
  public int foo(Array a) { blah}
}
```

Abstract Classes and Hiding



Smalltalk VM on startup informs Filename of the correct concrete class for the current platform

‘foo’ asFilename
Filename named: ‘foo’

Create an instance of the correct concrete Filename class

Platform Independence Aside

End of line Characters

Mac, PC and Unix have different end of line characters

When you read a file:

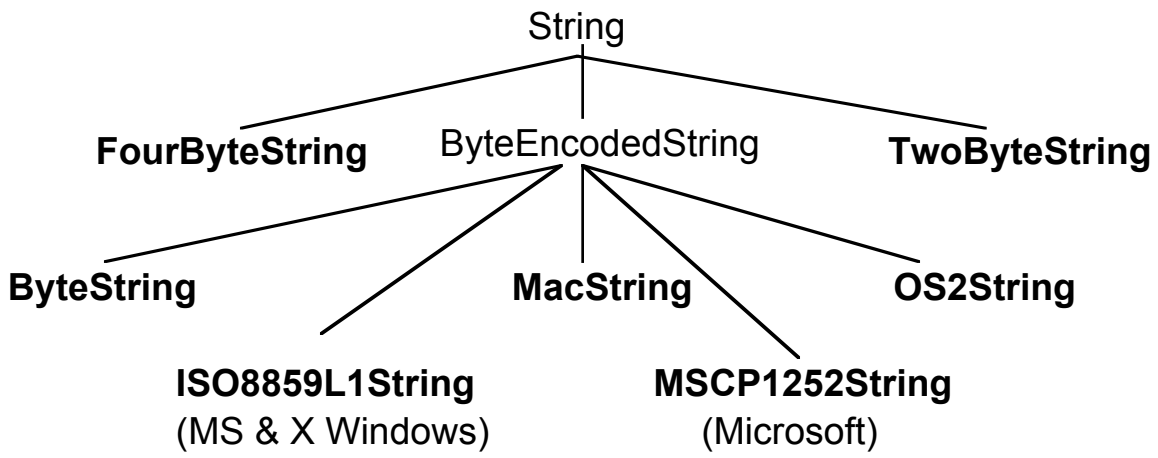
- Smalltalk converts the platform's end of line character to cr

When you write a file

- Smalltalk converts cr to the platform's end of line character

Same code
- Works on all three platforms
- Produces files with the correct end of line character

Abstract Classes and Hiding



String is an abstract class

String new
- Does not create a string object
- Creates an instance of a subclass
- Appears to create a String object

String subclasses
- Don’t add new methods
- Provide specific implementations

Strings Continued

```
| a |
a :=String new.
a class.          "returns ByteString"

| b |
b :=(String with: (Character value: 3585)) "3585 is Thai character".
b class           "returns TwoByteString"

| c |
c := String with: $a.
c class.          “returns ByteString”
c at: 1 put:  (Character value: 3585).
c class           “returns TwoByteString”
```

To learn about character encodings read:
http://www.joelonsoftware.com/articles/Unicode.html

become: Smalltalk Magic

```
| c |
c := String with: $a.
c class.          "returns ByteString"
c at: 1 put:  (Character value: 3585).
c class           "returns TwoByteString"
```

How did c change class?

a become: b

- Change all references to 'a' to reference 'b'
- Change all references to 'b' to reference 'a'
- 'a' basically becomes 'b' and 'b' becomes 'a'

String Transformation without become?

Use composition

String has instance variable that holds real string

String forwards messages to the real string

String can replace the real string with a different object

```
Smalltalk.Core defineClass: #String
   superclass: #{Core.CharacterArray}
   indexedType: #none
   private: false
   instanceVariableNames: 'realString'
   classInstanceVariableNames: ''
   imports: ''
   category: 'Collections-Text'

size
   ^realString size

at: anInteger
   ^realString at: anInteger

at: anInteger put: aCharacter
   aCharacter value > 256
      ifTrue: [realString := realString atTwoByteString].

   realString at: anInteger put: aCharacter.
```

Inheritance

What should I use as a super class?

- A has a B

Indicates that an instance variable of A is an instance of B

- A is a B
- A is a type of B

Indicates that A is a subclass of B

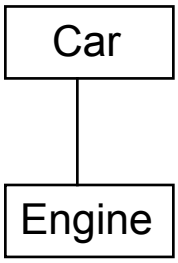
A car has an engine, so car object contains an engine object

A BinarySearchTree has nodes, so it has instance variables left and right

A WordStream is a type of ReadStream so it is a subclass of ReadStream

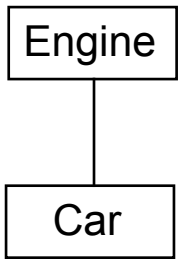
Common Mistakes

Engine Subclass of Car



Using a has-a relation for inheritance

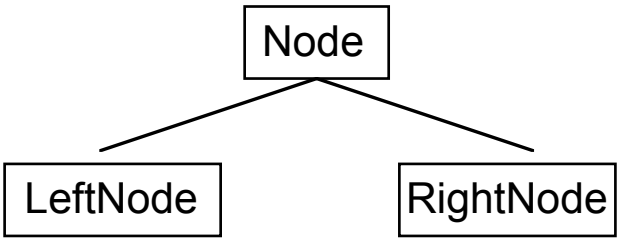
Car subclass of Engine



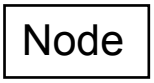
“I need access to engine methods in the car class and now I have it.”

Roles Verses Classes

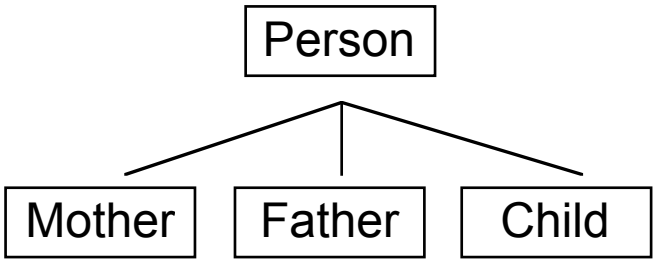
2.11 Be sure the abstractions you model are classes and not simply the roles objects play




```
BinarySearchTree>>initialize
    left := LeftNode new.
    right := RightNode new.
```




```
BinarySearchTree>>initialize
    left := Node new.
    right := Node new.
```




```
initialize
   mother := Mother new.
   father := Father new.
etc.
```


```
initialize
   mother := Person new.
   father := Person new.
etc.
```

What to Test

Everything that could possibly break

How often do accessor methods have errors?

```
Node>>value
   ^value
```

How many errors did your WordStream>>next method have?

Some Guidelines

Test values

- Inside valid range
- Outside valid range
- On the boundary between valid/invalid

GUIs are very hard to test

- Keep GUI layer very thin
- Unit test program behind the GUI, not the GUI

Common Things that Programs Handle Incorrectly

Adapted with permission from "A Short Catalog of Test Ideas" by Brian Marick, http://www.testing.com/writings.html

Strings

Test using empty String

Collections

Test using:
- Empty Collection
- Collection with one element
- Collection with duplicate elements
- Collections with maximum possible size

Numbers

Test using:
- Zero
- The smallest number
- Just below the smallest number
- The largest number
- Just above the largest number

Do we need to test all methods?

```
Character>>isWordSeparator
   ^self isSeparator | (#($, $. $; $' $" $? $!) includes: self)

Character>>dollarValue
   self isAlphabetic ifFalse:[^0].
   ^self asLowercase asInteger - $a asInteger + 1

String>>dollarValue
   ^self inject: 0 into: [:subTotal :next | subTotal + next dollarValue]

String>>words
   ^self runsFailing: [:each | each isWordSeparator]

String>>dollarWords
   ^self words select: [:each | each dollarValue = 100]
```

Can we just test String>>dollarWords?

If String>>dollarWords works correctly then all the other methods are highly likely to also work correctly

Benefit
- Fewer tests to write
- Fewer tests to maintain
- More time to write code and write tests that matter

Drawbacks
- Harder to find errors
- Harder to test important cases

Can we just test

String>>dollarValue
String>>words

If these methods work what could be wrong with String>>dollarWords?